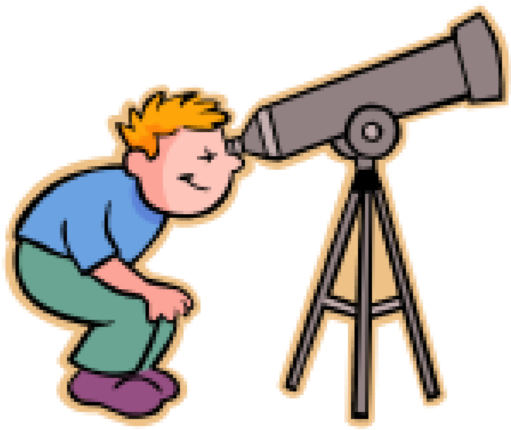
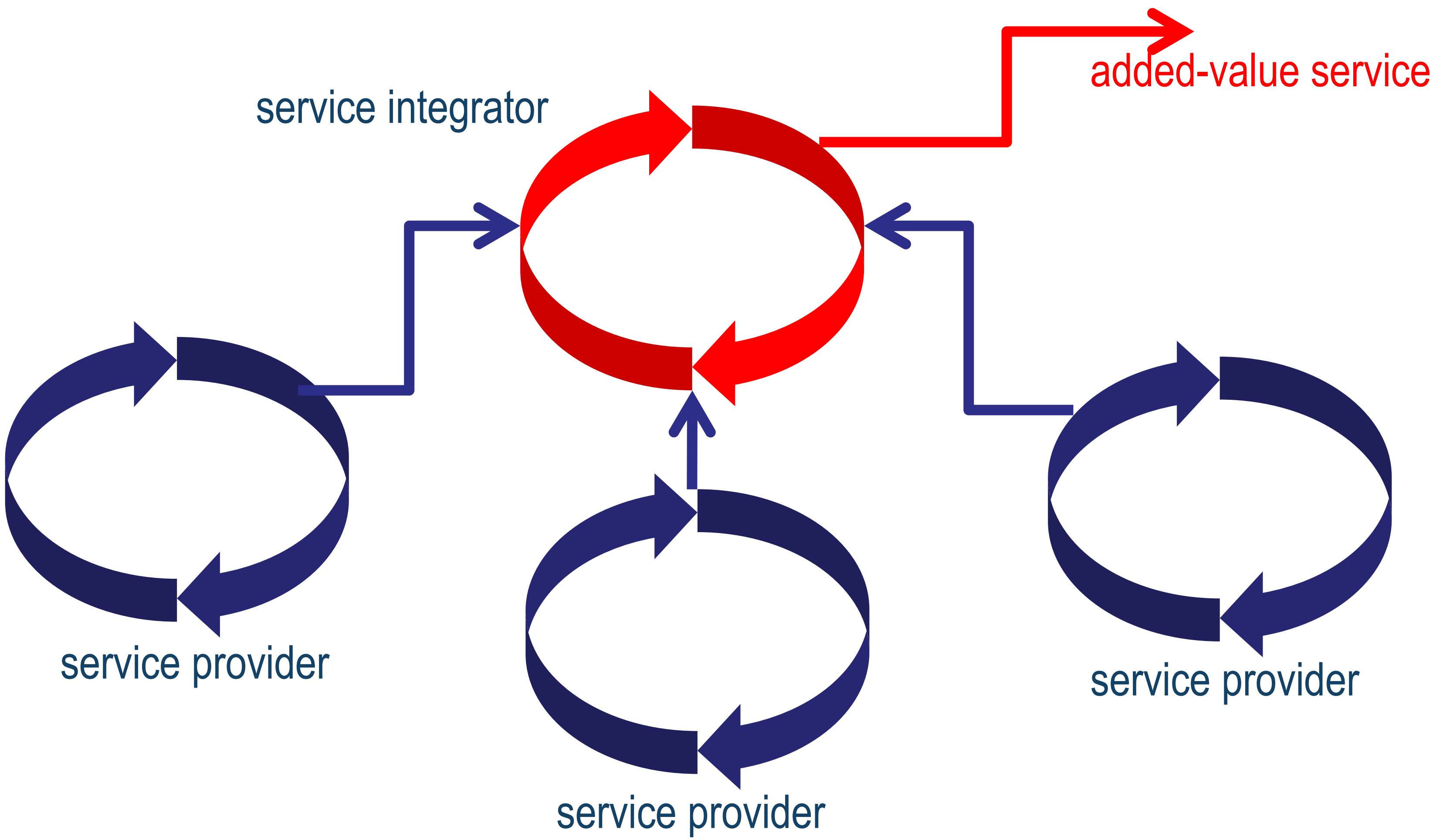
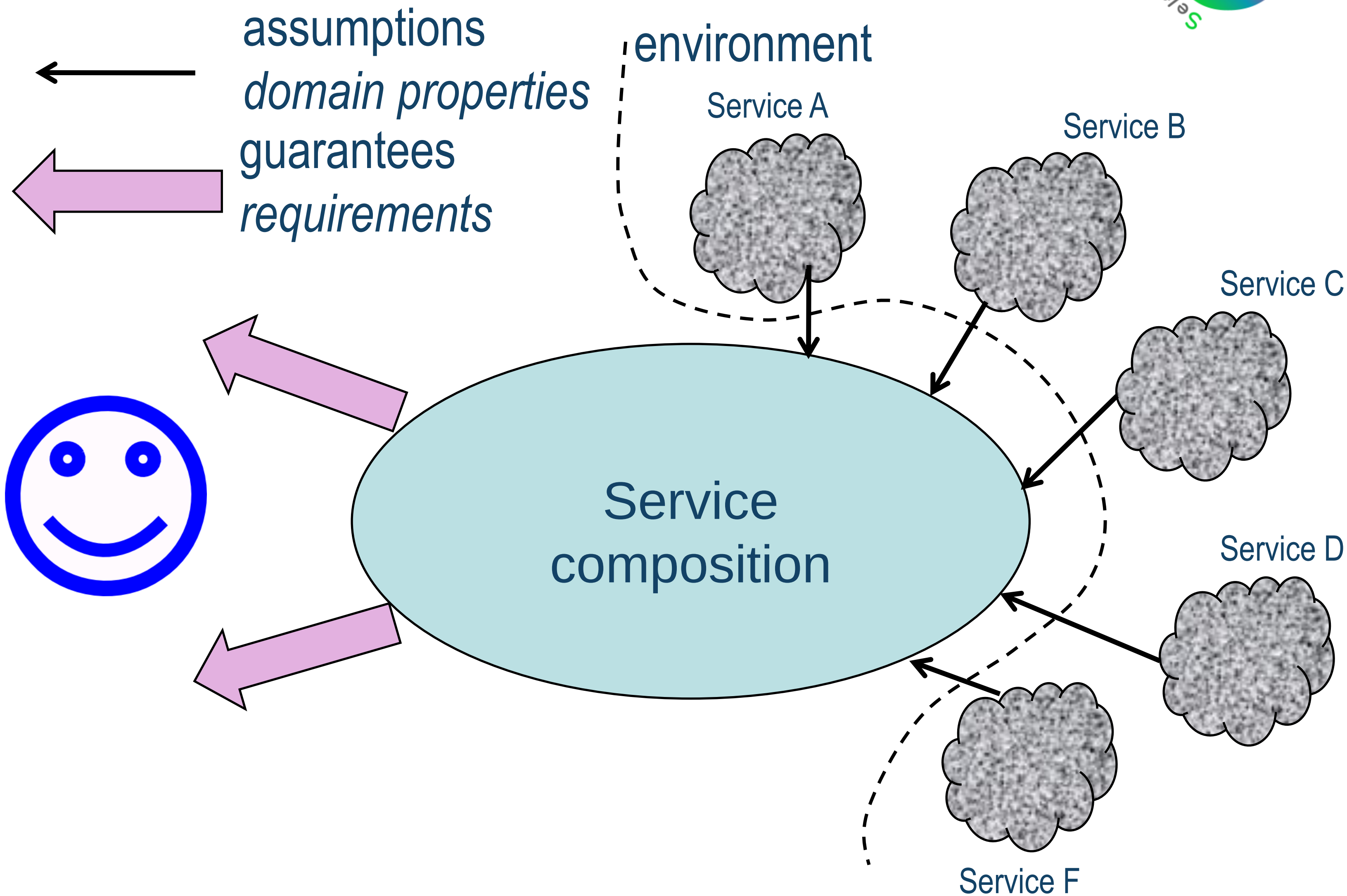


From requirements to specification and (continuous) verification (Part 1 -- SAVVY-WS)

Our setting


added-value service
service integrator
service provider
service provider
service provider

Our setting



assumptions
domain properties
guarantees
requirements
environment
Service A
Service B
Service C
Service D
Service F
Service composition

Open world: the problem

- External services may evolve autonomously
- The assumptions made at design-time may be later invalidated
- What can be done at design-time?
- What needs to be done at run-time?

SAVVY-WS

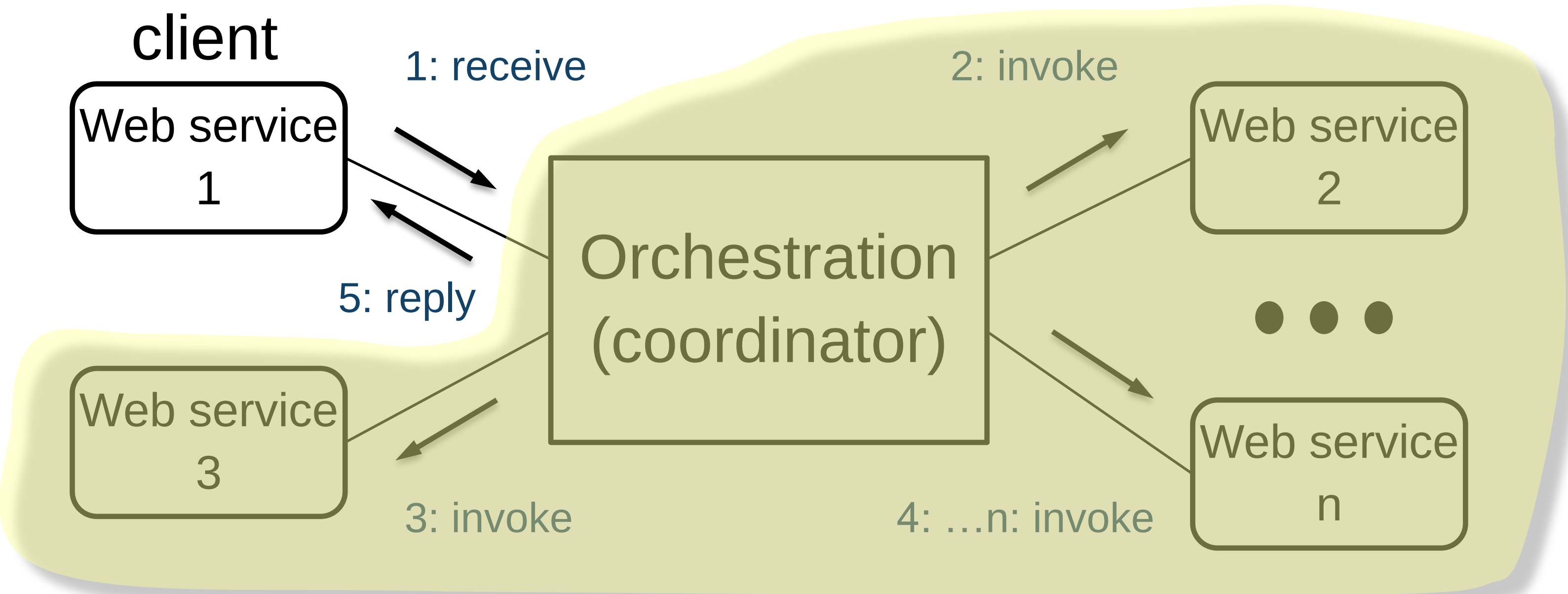
- **S**ervice **A**nalysis, **V**erification and **V**alidation methodolog**Y** for Web Services
  - It supports the development of **verified composite services**, built as BPEL workflows
- Compositions are guaranteed to satisfy certain global correctness properties
- External services are assumed to be known at the level of the interface (**abstract services**) and their assumed behavior is specified as we will describe
  - any **concrete service** that offers a “compatible” interface may be later bound

Conceptual approach

- An **assumption-promise** based approach
  - a service integrator **assumes** that the external services used in the composition satisfy their stated specification
  - under this assumption, the system is designed to **promise** a certain service to its clients
- But since the external services may deviate wrt to their stated specification
  - a **monitor** does run-time verification
  - suitable **reactions** may be activated
    - *reactions ignored in this presentation*

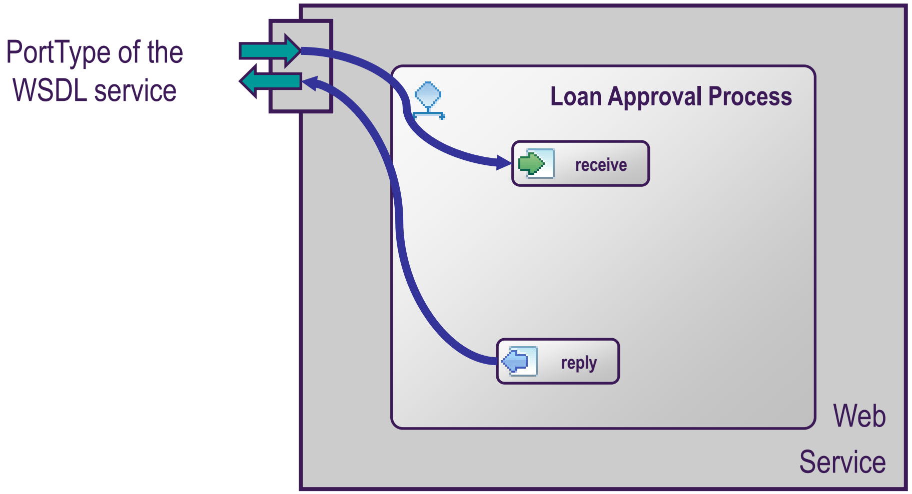
BPEL—Business Process Execution Language

- Supports the definition of Business Processes (BPs) which use external Web Services
- BPs *coordinate* (***orchestrate***) external Web services
- A BPEL BP can be seen in turn as a service



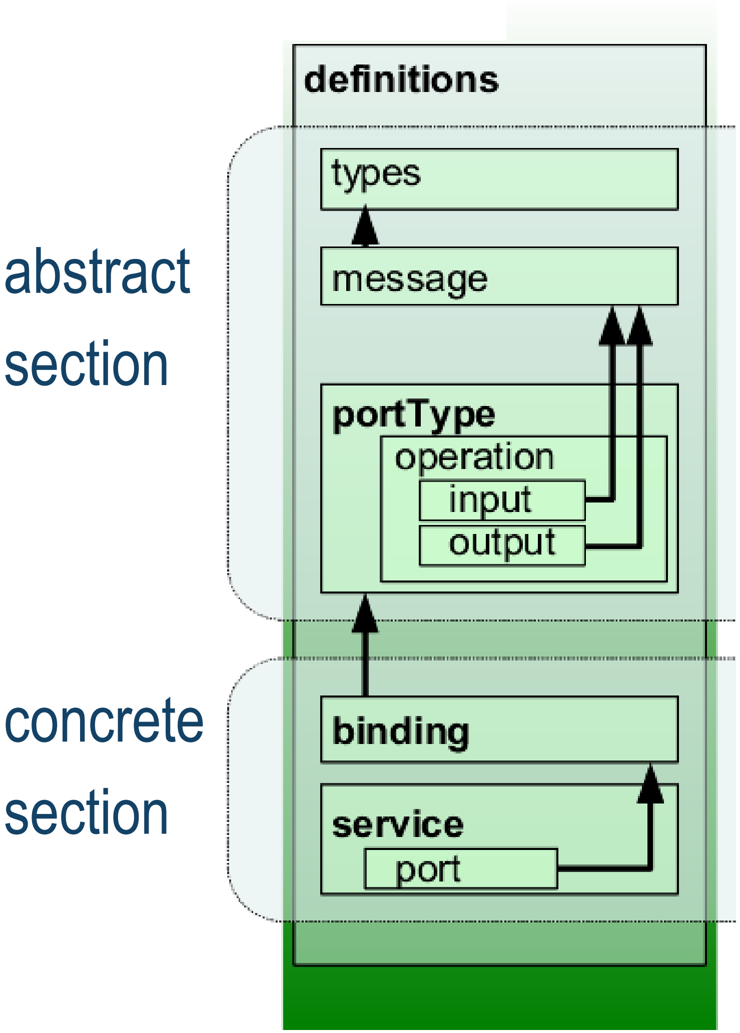
BPEL and WSDL

- WSDL Web Service Description Language
  - syntactic description
- BPEL processes are exposed as services through a WSDL interface
  - message exchanges depend on the defined WSDL operations

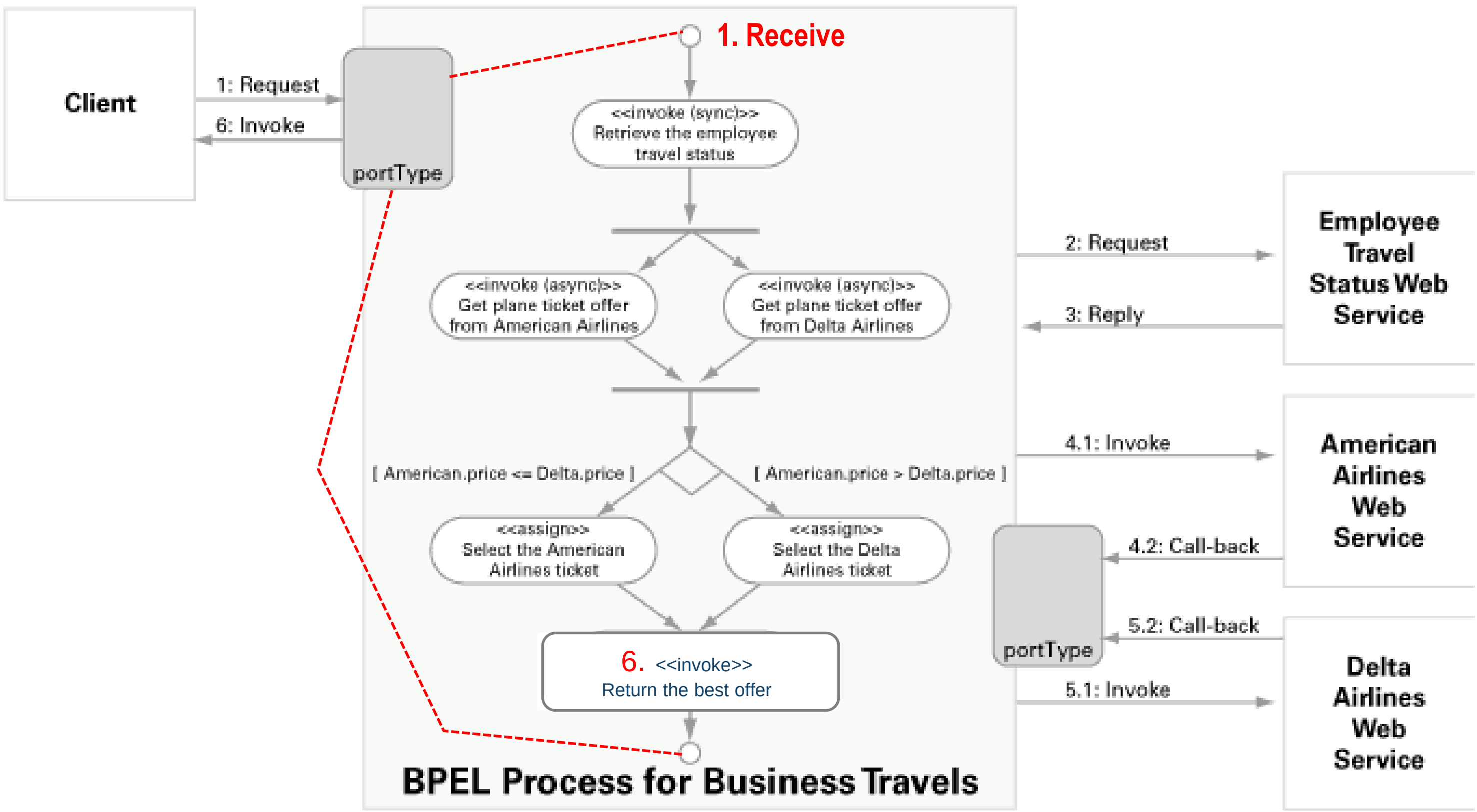


WSDL

- Describes the interface of a service in terms of operations and parameters
- Contains definition of message types
- The description is an XML document



Example: a preview



Self Managing Situated Computing

BPEL Basic Activities

```
<invoke partnerLink="..." portType="..." operation="..."
        inputVariable="..." outputVariable="..."/>
   <!-- process invokes an operation on a partner:    -->

<receive partnerLink="..." portType="..." operation="..."
         variable="..." [createInstance="..."]/>
   <!-- process receives invocation from a partner:    -->
<reply partnerLink="..." portType="..." operation="..."
       variable="..."/>
   <!-- process sends reply message in partner invocation:    -->
<assign>
   <copy>
     <from variable="..."/> <to variable="..."/>
   </copy>+
</assign>
   <!– Data assignment between variables    -->
```

More Basic Activities

```
<throw faultName="..." faultVariable="... "/>
```

<!-- process signals an internal fault -->

```
<terminate />
```

<!– terminates the process execution -->

```
<wait (for="..." | until="...")/>
```

<!-- process execution is delayed for a certain period of time or until a certain deadline is reached -->

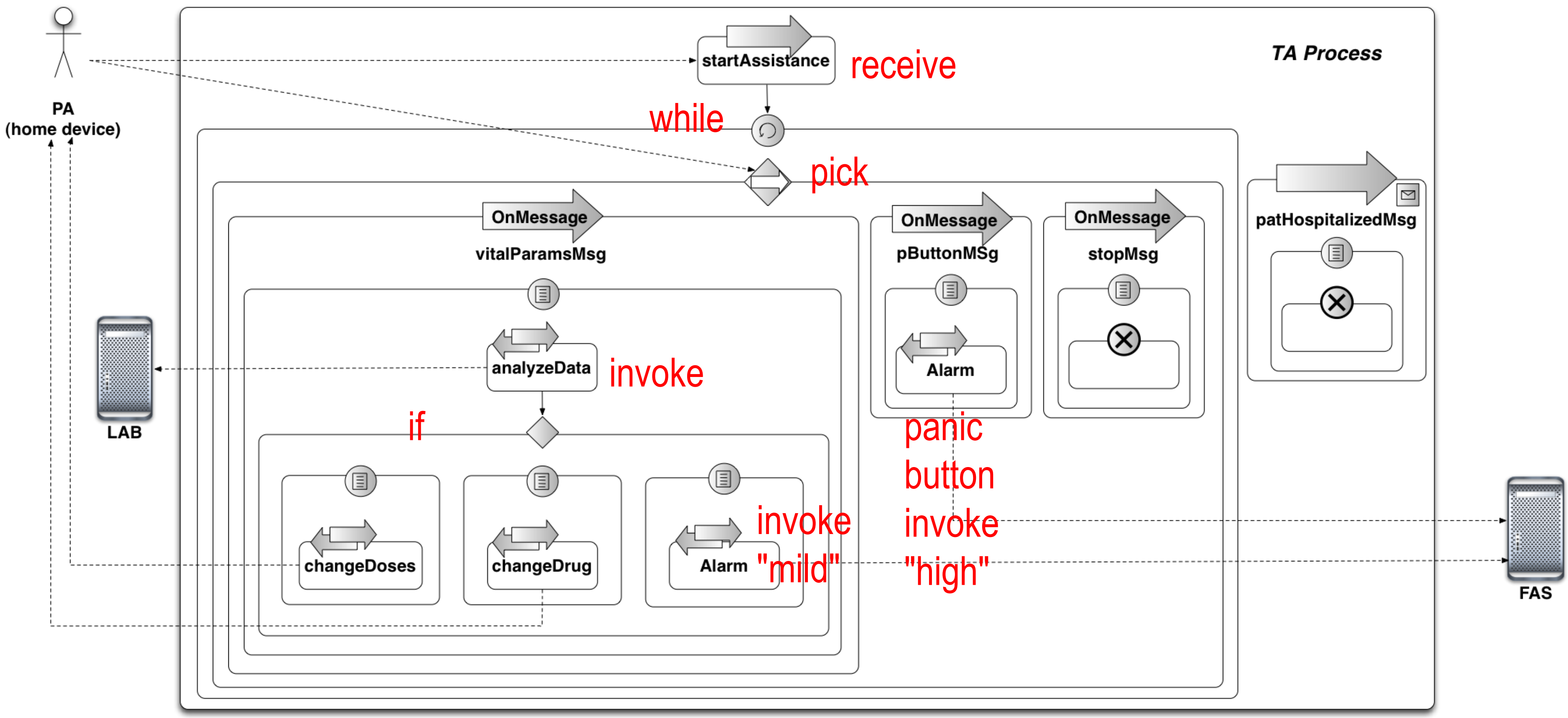
```
<empty />
```

<!– Do nothing; a convenience element -->

Variables

- Necessary to maintain the process state
- Their types can be:
  - WSDL message
  - XML type
  - XML Schema element
- Contents of (inbound and outbound) messages are stored in variables

The TeleAssistance (*TA*) Process



Assumed properties

- LabServiceTime—Lab

  after sending the patient's data to the lab, a reply is received within 1 hour
- FASConfirmHospitalization—FA Squad

  if the FAS is invoked three times over a week, with a "High" severity level for a certain patient, within one day a notification is received that the patient has been hospitalized

Promised properties

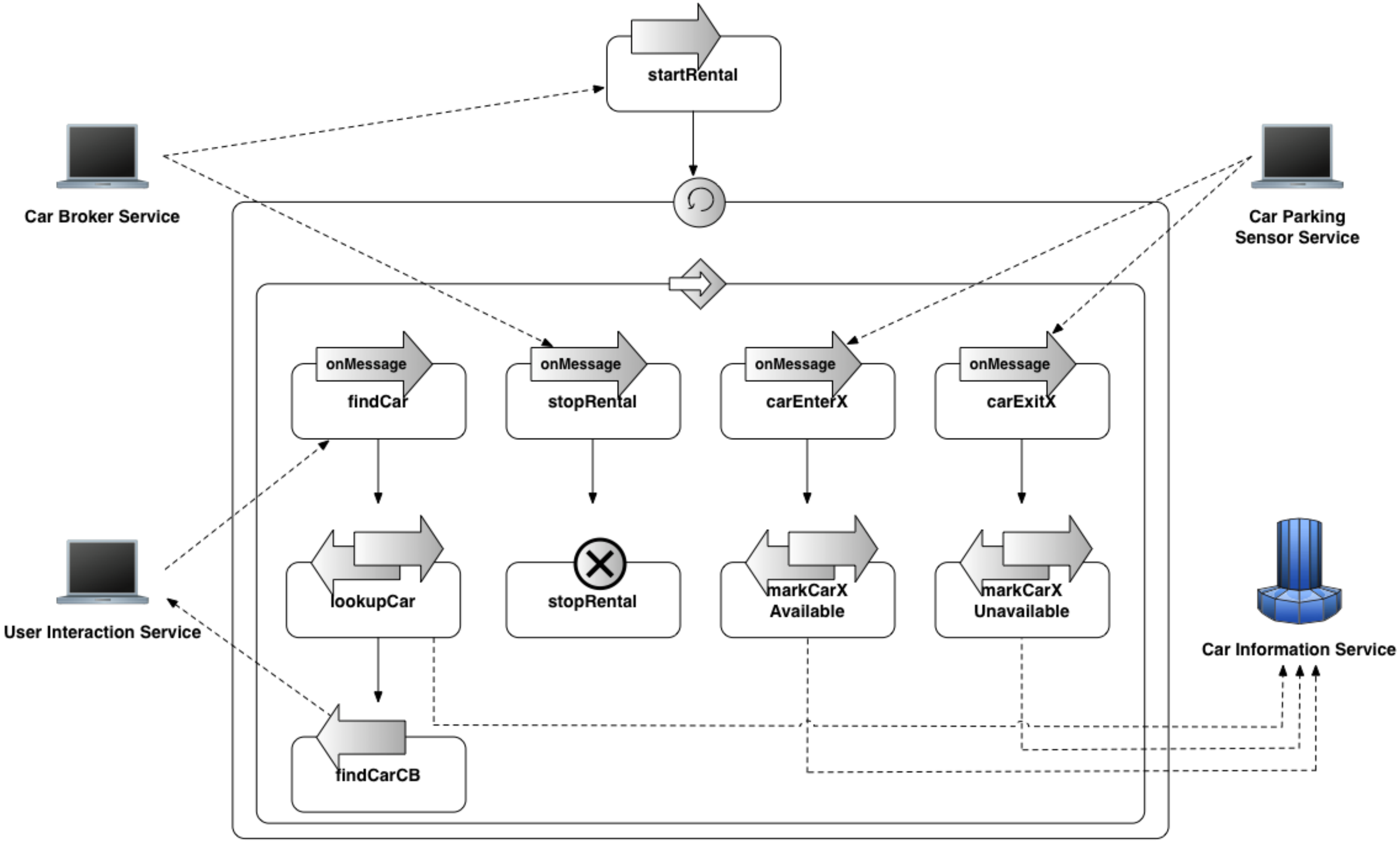
- FASInvokeMildAlarm

  after receiving a message from the LAB indicating that an alarm must be issued to the FAS, the TA process must send a "Mild" alarm notification to the FAS service within four hours

- MDCheckUp

  if a certain patient pushes the pButtonMsg three times during a time span of a week, the patient must be hospitalized within one day

Car rental process



Sample properties

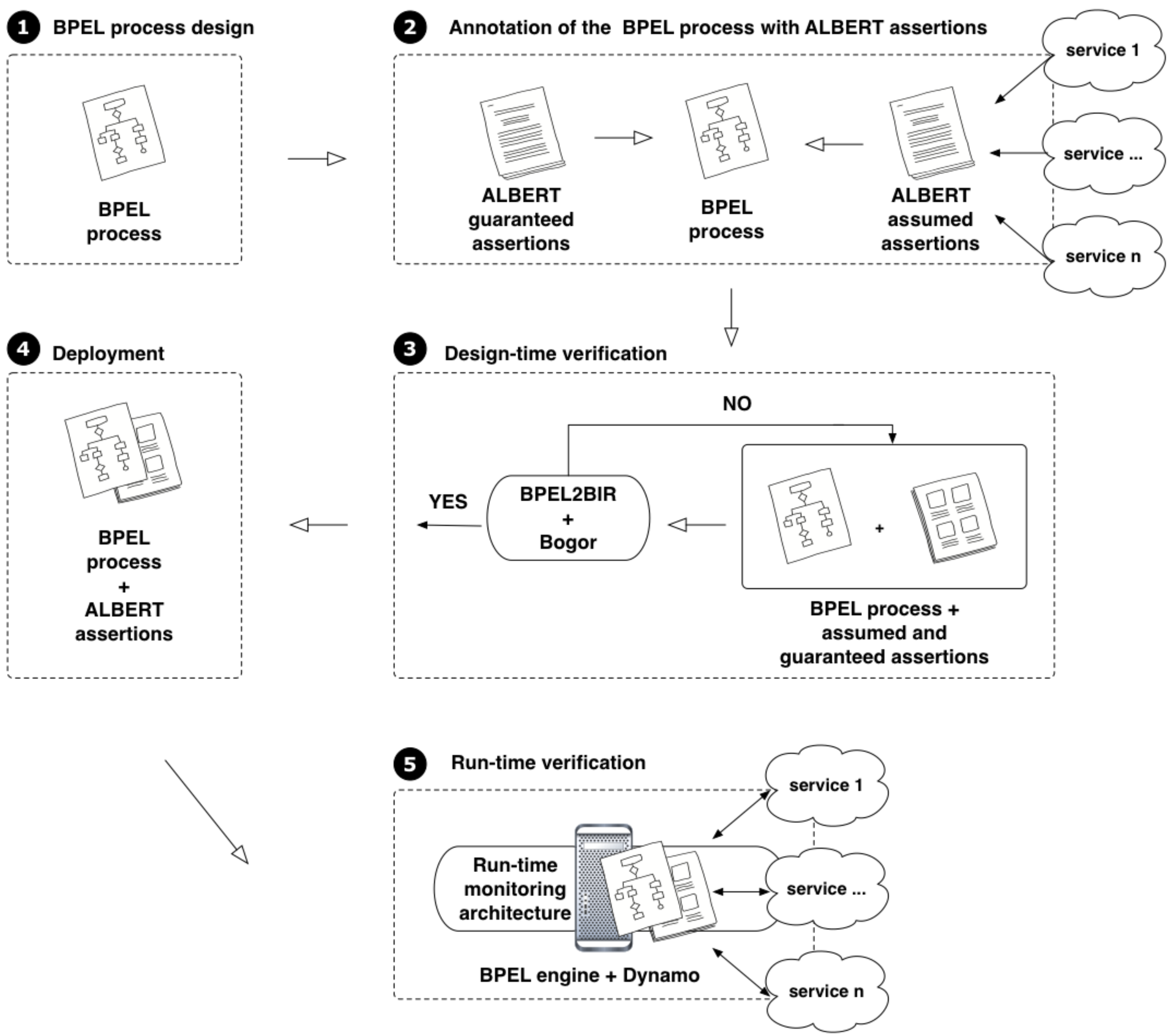
- ParkingInOut (AP)—car parking sensor service

  between two events signaling that a car exits the parking, an event signaling entrance for the same car must occur
- RentCar (PP)

  if a car enters the parking and does not exit until a client requests it for renting, then the request must succeed

Two kinds of verification, one language

- ALBERT (Assertion Language for BPEL pRocess inTeractions)
  - can express assumptions and promises
- Can be used for two kinds of verification
  - **design-time** (model checking)
    - promised properties are satisfied by the workflow under the assumption that assumed properties hold
  - **run-time** (monitoring+ run-time verification)
    - checks if assumptions are valid
      - services satisfy their promises


1 BPEL process design
BPEL process
2 Annotation of the BPEL process with ALBERT assertions
ALBERT guaranteed assertions
BPEL process
ALBERT assumed assertions
service 1
service ...
service n
3 Design-time verification
NO
YES
BPEL2BIR + Bogor
BPEL process + assumed and guaranteed assertions
4 Deployment
BPEL process + ALBERT assertions
5 Run-time verification
Run-time monitoring architecture
BPEL engine + Dynamo
service 1
service ...
service n

ALBERT

- A linear temporal logic language
- **Variables** correspond to BPEL variables
- **State** a triple (V, I, t), where
  - V is a set of <var, val> pairs
  - I is a location in the workflow: set of labels
  - t is the time at which the state is generated
- State changes are associated with location counter change in the workflow
  - internal activities (assign)
  - interactions with the world

ALBERT in a nutshell

- It predicates on variables
- Classical boolean operators and quantifications
- Event predicate
  - OnEvent(*XXX*) true in a state if event *XXX* occurs in that state; e.g., *onEvent(invoke_XXX)*
    - true in a state if the service XXX is invoked in that state
- Future Temporal Operators
  - Becomes, Until, Within
- Functions
  - elapsed, past, count, …

Self Managing Situated Computing

ALBERT—syntax

$\phi ::= \neg\phi \mid \phi \wedge \phi \mid \forall var \phi \mid Becomes(\phi) \mid Until(\phi, \phi) \mid Within(\phi, K) \mid \psi \text{ relop } \psi \mid onEvent(\mu)$

$\psi ::= var \mid \psi \text{ arop } \psi \mid const \mid past(\psi, onEvent(\mu)) \mid count(\phi, K) \mid fun(\psi, K) \mid fun(\psi, onEvent(\mu), K) \mid elapsed(onEvent(\mu))$

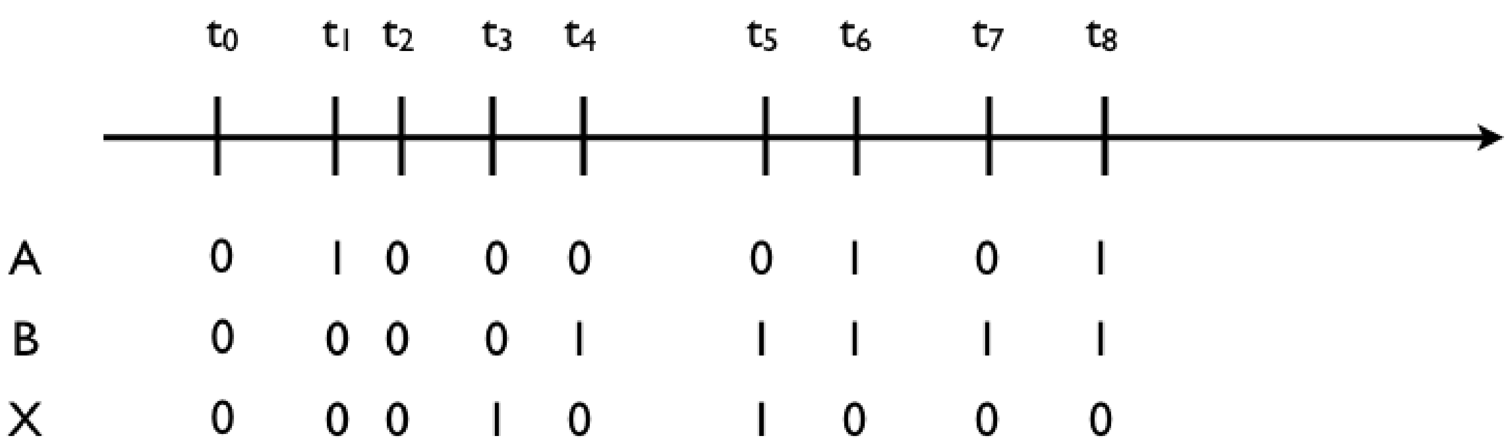
$relop ::= < \mid \leq \mid = \mid \geq \mid >$

$arop ::= + \mid - \mid \times \mid \div$

$fun ::= sum \mid avg \mid min \mid max \mid \ldots$

ALBERT—semantics

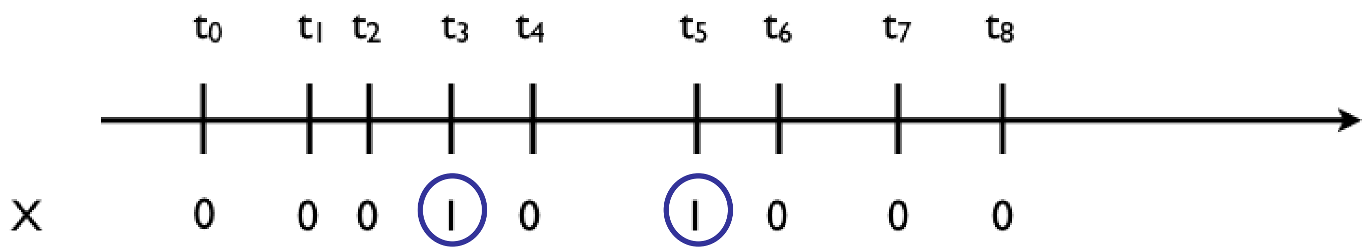
- Semantics is explained by referring to sequences of time-stamped states of the BPEL process (timed state word)
  - an infinite sequence s_0, s_1, s_2, ..., where each s_i is a state (V_i, I_i, t_i)
    - the sequence is strictly monotonic (time consuming operation occurs in a transition)



Becomes and Until

Becomes(X)

true when evaluated in t_3 and t_5



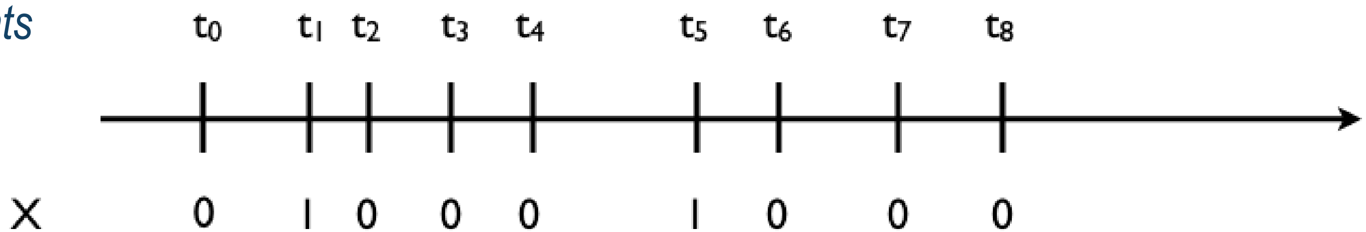
Until(A,B)

true when evaluated in t_2 (and also in t_1)

Within

Within (X, K)

- *X true within K instants*
- true when
- evaluated in t_2
- with $t_5 - t_2 \leq K$

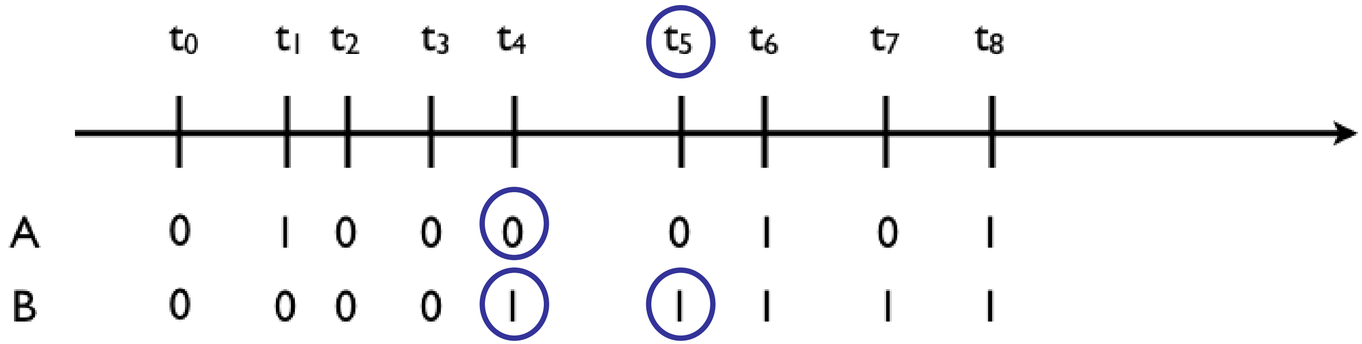


past and elapsed

past(A,onEvent(B))

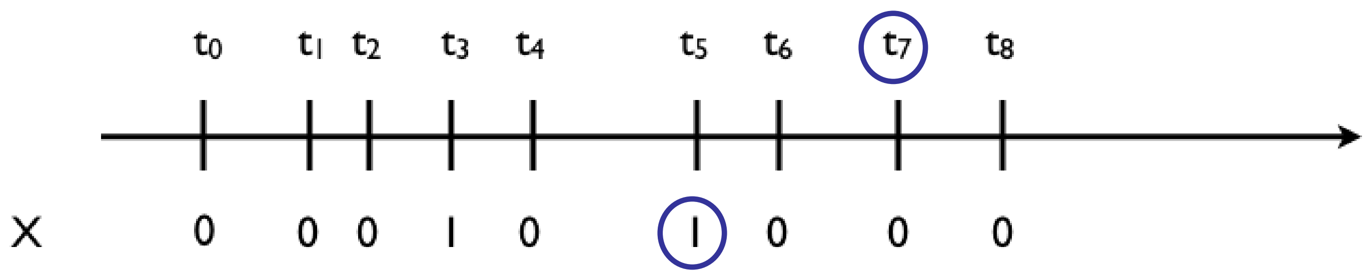
= 0 when evaluated in t_5 and in t_6

= 1 when evaluated in t_7

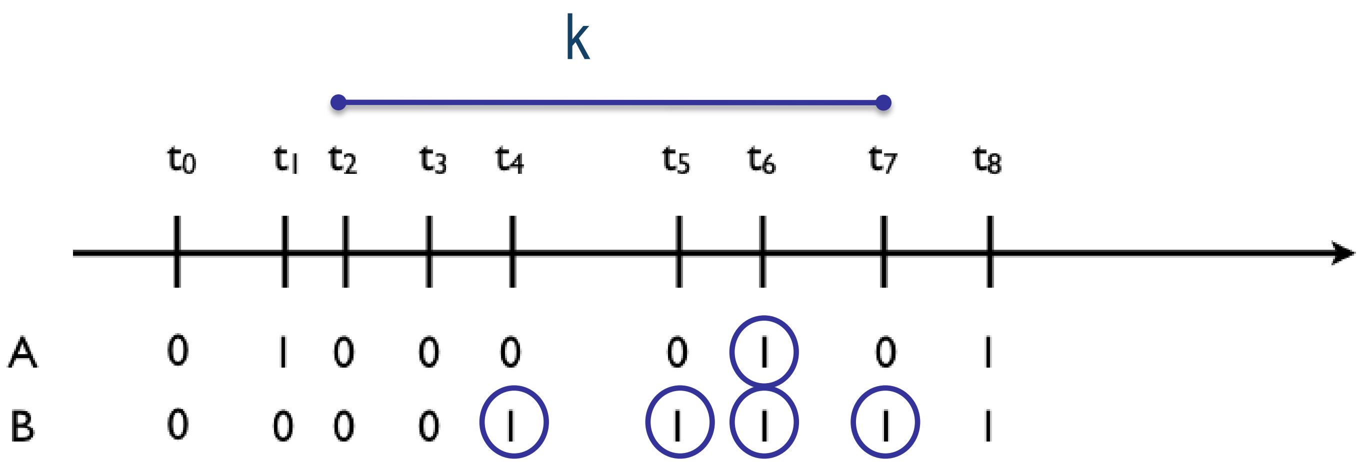


elapsed(onEvent(X))

= t_7-t_5 when evaluated in t_7



count



- In t_7 (with t_7-$t_2 \leq k$ and t_7-$t_1 > k$):
  - *count(A, k)* = 1
  - *Becomes(count(B, k) = 4)* is true

ALBERT properties

- Properties are in the form

  it is always true that XXX holds

  that is, they are process invariants
- In Albert, they have the form

  Always XXX

  - which stands for ***XXX and not (true Until not XXX)***
- For convention, the outer Always is omitted

Self Managing Situated Computing

Useful derived operators

- We saw **Always** (also $\Box$)
- **Eventually** (also $\Diamond$)
  - $\Diamond X$ defined as *true Until X*
- *When(X, Y)* when A is true in the future, B is also true
  - *When(X, Y)* defined as $\Diamond X \rightarrow (\neg X \; Until \; X \wedge Y)$

Using ALBERT (TeleAssistance)

- LabServiceTime (AP)

  after sending the patient's data to the lab, a reply is received within 1 hour

onEvent(invoke_AnalyzeData) →
    Within(*onEvent*(receive_AnalyzeData), 60)

Using ALBERT (TA cont.)

- AverageLabServiceTime (AP)

  the average response time of requests to analyze data completed in past 10 hrs should be less than 45 min.

avg(elapsed(onEvent(invoke_AnalyzeData)),
onEvent(receive_AnalyzeData), 600) <= 45

Using ALBERT (TA cont.)

- FASInvokeMildAlarm (PP)

  after receiving a message from the LAB indicating that an alarm must be issued to the FAS, the TA process must send a "Mild" alarm notification to the FAS service within four hours

onEvent(receive_AnalyzeData) ∧
$analysisResult/suggestion = ‘sendAlarm’)
→
Within(*onEvent*(alarmNotif) ∧$alarmNotif/level= ‘mild’),240)

Using ALBERT (TA cont.)

- MDCheckUp (PP)

  if a certain patient sends the pButtonMsg three times during a time span of a week, the patient must be hospitalized within one day

$$\forall x\ (\textit{Becomes}\ (\textit{count}(\text{onEventl}(\text{pButtonMsg}) \wedge \$\text{alarmNotif/pId}=x,\ 10080)\ =3) \rightarrow \textit{Within}(\textit{onEvent}(\text{patHospitalized}) \wedge \$\text{patHospitalized/pID}=x,\ 1440))$$

Using ALBERT (CarRental)

- ParkingInOut (AP)

  between two events signaling that a car exits the parking, an event signaling entrance for the same car must occur

∀ x ((*onEvent*(carExit) ∧$carExit/carID=x) →
Until(*not* (*onEvent*(carExit) ∧$carExit/carID=x),
*onEvent*l(carEnter) ∧$carEnter/carID=x))

Using ALBERT (CR cont.)

Self Managing Situated Computing

- CISUpdate (AP)

  if a car is marked available in the Car InfoSyst and it is not marked unavailable until a lookUpCar is performed, then lookup for that car must show that the car is available

∀ x (*onEvent*(receive_MarkAvail) ∧ $carInfo/carID=x ∧
Until (*not* (*onEvent*(receive_MarkUnavail)∧$carInfo/carID=x),
onEvent(invoke_LookUp) ∧ $carInfo/carId=x)
→
When(*onEvent*(invoke_LookUp) ∧ $carInfo/carId=x,
Eventually((*onEvent*(receive_LookUp) ∧
$carInfo/carId=x∧$qRes/res!="no")))

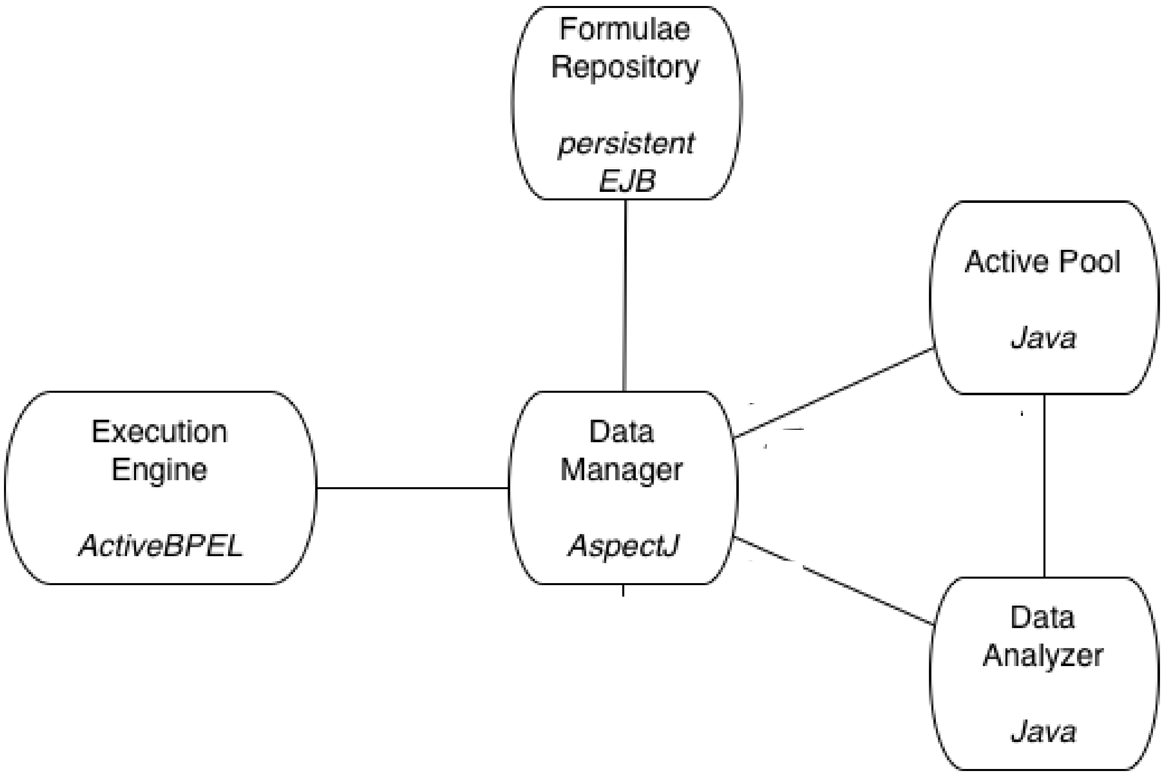
Using ALBERT (CR cont.)

- RentCar (PP)

  if a car enters the parking and does not exit until a client requests it for renting, then the request must succeed

∀ x (*onEvent*(carEnter) ∧ $carEnter/carID=x ∧
Until (*not* (*onEvent*(carExit) ∧ $carEnter/carID=x),
onEvent(invoke_FindCar) ∧ $carInfo/carId=x))
→
When(*onEvent*(invoke_FindCar) ∧ $carInfo/id=x,
Eventually((*onEvent*(invoke_FindCarCB) ∧ $carInfo/carId=x
∧ $queryResult/res!="no")))

Execution and monitoring



Properties of run-time verification

- Size of the state history kept in the ActivePool
  - Maximum among:
    - maximum nesting level of *past* functions
    - 1 (if there is at least a *Becomes* predicate)
    - maximum number of states needed for the various *count* and *fun* time windows
- Number of threads required for the verification
  - 1, for each *Until* (sub)formula
  - number of states in the sequence of process states that may occur in a time interval long K, for each *Within(A,K)* (sub)formula